Developing Efficient Code for Decoding AIS Sentences into Useful Datasets

Kiki Beumer

International Association of Marine Aids to Navigation and Lighthouse Authorities

CY Tech Cergy Paris Université

July 11, 2024

Contents

1 Introduction

The purpose of this code is to decode AIS data transmitted using the NMEA 0183 standard, developed and maintained by the National Marine Electronics Association (NMEA) to establish interface standards for marine electronic equipment.

To extract useful information from these AIS sentences, we decode them into an understandable dataframe. Existing AIS decoding codes often require significant time; for example, decoding 1 million sentences can take approximately 3-4 hours. In contrast, this code achieves the same task in significantly less time, approximately 5-9 minutes for decoding approximately 6.5 million sentences.

This decoding process is simplified in that it focuses solely on extraction without filtering or encoding data as needed, which may be considered a limitation. The code serves as a foundational framework intended for future expansion and enhancement. Additional functionalities can be incorporated as necessary.

This report provides an overview of the code's functions, facilitating a quick understanding of its framework. It aims to empower users to extend and customize the code to meet specific requirements.

2 Decoding AIS Data

AIS messages are commonly transmitted using the NMEA 0183 standard, developed and maintained by the National Marine Electronics Association (NMEA) to establish interface standards for marine electronic equipment. NMEA 0183 is a standard framework for exchanging marine instrument data between various onboard equipment.
An example of an AIS sentence is as follows:

```
!AIVDM,2,1,3,B,55P5TL01VIaAL@7WKO@mBplU@<PDhh000000001S;AJ::4A80?4i@E53,0*3E
!AIVDM,2,2,3,B,1@0000000000000,2*55
```

Figure 1: Example of a multi fragment sentence

Field 1, Format: !AIVDM, identifies this as an AIVDM packet. Other packets are i.a. AIVDO, BSVDM, BSVDO.

Field 2, message count: Total number of messages, sometimes AIS messages are split over several messages due to size limitation.

Field 3, message number: The fragment number of this sentence. It will be one-based. A sentence with a fragment count of 1 and a fragment number of 1 is complete in itself.

Field 4, sequence ID: The message ID if message count is larger than 1.

Field 5, radio channel code: AIS uses the high side of the duplex from two VHF radio channels: AIS Channel A is 161.975Mhz (87B); AIS Channel B is 162.025Mhz (88B).

Field 6, payload: This is the AIS data itself encoded in six bit ASCII. Information such as MMSI number, navigation status, longitude, latitude and speed.

Field 7, size: The number of bits required to fill the data.

Field 8, checksum: The checksum is needed to verify sentence integrity.

3 Code Overview

This code contains several functions, each responsible for decoding a different part of the AIS sentence payload. These functions either format the data correctly or assign a value to the corresponding information.

The process begins by verifying the checksum of the AIS sentence to ensure data integrity. Next, all relevant fields are extracted from the sentence, such as the packet type (field 1) and radio channel (field 5). After this, the payload is extracted, decoded, and a binary string is returned. Based on the message type, specific sections of this binary string are extracted.

From this binary string, information such as the rate of turn, longitude, speed over ground, and other pertinent data is retrieved. Additional functions are then called to interpret this decoded data, providing it with meaningful context.

4 Detailed Explanation

- Purpose: Convert longitude from thousandths of a minute to degrees, minutes, and seconds.
- Parameters: value (integer representing longitude in 1/10000 minutes).
- Return Value: A string describing the longitude in DMS format.
- Algorithm: Explain the logic and calculations performed.

4.1 Main

The main script imports pandas to display the data as a dataframe. The data is imported from a text file, with sentences extracted line by line. The `decode_ais_nmea` function is called for each sentence, and the decoded messages are stored in a list. Finally, this list is displayed as a dataframe.

```
import pandas as pd

# Initialize a list
decoded_messages = []

# Open the file in read mode
with open("Dataset001.ais.txt", 'r') as file:
    for line in file:
        nmea_sentence = line.strip()
```

```python

        decoded_message = decode_ais_nmea(nmea_sentence)
        decoded_messages.append(decoded_message)

# Create a DataFrame from the list of decoded messages
df = pd.DataFrame(decoded_messages)
df
```

Listing 1: Python code for main

4.2 decode_ais_nmea Function

This function verifies the integrity of the checksum. The different fields described previously are stored in a list. From this list the payload is extracted and converted to a binary string. `decode_ais_message` finally decodes the binary string into meaningful information.

```python
def decode_ais_nmea(nmea_sentence):
    if not verify_checksum(nmea_sentence):
        return {'Error': 'Checksum mismatch'}

    fields = nmea_sentence.split(',')
    payload = fields[5]
    binary_payload = decode_payload(payload)
    decoded_message = decode_ais_message(binary_payload, fields)

    return decoded_message
```

Listing 2: Python code for `decode_ais_nmea` function

4.3 verify_checksum Function

As mentioned previously, the checksum verifies the accuracy of the AIS sentence. In this text file, a timestamp appears before the AIS sentence, so the timestamp is extracted first, followed by the checksum and then the '!'. The remaining part of the sentence is iterated over, performing the XOR operation to calculate the checksum. This function returns whether the expected and calculated checksums are equal.

```python
def verify_checksum(nmea_sentence):
    """Verify the NMEA sentence checksum."""
    time, ais_sentence = nmea_sentence.split(' ', 1)
    nmea_data, checksum = ais_sentence.split('*')
    nmea_data = nmea_data.lstrip('!')

    calc_checksum = 0
    for char in nmea_data:
        calc_checksum ^= ord(char) #returns integer unicode(subset of
    ASCII)

    return int(checksum, 16) == calc_checksum
```

Listing 3: Python code for `verify_checksum` function

4.4 decode_payload Function

The sole goal of this function is to convert the six bit ASCII payload into binary. The format(value, '06b') statement is used to format an integer value as a binary string.

```python
def decode_payload(payload):
    six_bit_ascii = {
        '0': 0, '1': 1, '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7,
        '8': 8, '9': 9, ':': 10, ';': 11, '<': 12, '=': 13, '>': 14, '?': 15,
        '@': 16, 'A': 17, 'B': 18, 'C': 19, 'D': 20, 'E': 21, 'F': 22, 'G': 23,
        'H': 24, 'I': 25, 'J': 26, 'K': 27, 'L': 28, 'M': 29, 'N': 30, 'O': 31,
        'P': 32, 'Q': 33, 'R': 34, 'S': 35, 'T': 36, 'U': 37, 'V': 38, 'W': 39,
        '`': 40, 'a': 41, 'b': 42, 'c': 43, 'd': 44, 'e': 45, 'f': 46, 'g': 47,
        'h': 48, 'i': 49, 'j': 50, 'k': 51, 'l': 52, 'm': 53, 'n': 54, 'o': 55,
        'p': 56, 'q': 57, 'r': 58, 's': 59, 't': 60, 'u': 61, 'v': 62, 'w': 63
    }

    binary_str = ''
    for char in payload:
        if char in six_bit_ascii:
            value = six_bit_ascii[char]
            binary_str += format(value, '06b')

    #Add else statement here if needed

    return binary_str
```

Listing 4: Python code for decode_payload function

4.5 decode_ais_message Function

The parameters of this function are the previously obtained binary string and the fields list. Based on the message type, the binary string is decoded. Not all variables are used for each message type, so the variables are first initialized to ensure they appear neatly in the dataset. Additional information is assigned to these variables as required. Before decoding, if len(binary_str) < n: checks whether the binary string meets the length requirements to avoid incorrect values. All decoded information is then returned.

```python
from datetime import datetime

def decode_ais_message1(binary_str, fields):

    #Columns for all message types
    message_type = int(binary_str[0:6], 2)
    repeat_ind = int(binary_str[6:8], 2)
    mmsi = int(binary_str[8:38], 2)
```

```
    time_format, packet_type = fields[0].split(' ', 1)
    channel = fields[4]

    #time
    time = datetime.strptime(time_format, "%Y-%m-%dT%H:%M:%S.%fZ")

    #Columns different per message type
    ais_version = 0
    imo = 0
    call_sign = 0
    vessel_name = 'NaN'
    ship_type = get_ship_type(0)
    a = 0
    b = 0
    c = 0
    d = 0
    eta = 'NaN'
    draught = 0
    destination = 'NaN'
    nav_status = 'NaN'
    rot = 'No turn info available'
    sog = 0
    cog = 'NaN'
    position_acc = 'NaN'
    long = 0
    lat = 0
    heading = 0
    radio_status = 0
    sotdma = 0
    pos_fix_epfd = get_position_fix_type(0)
    maneuver = get_maneuver_ind(0)

###############type 1,2,3 ###############
    if messag\_type in [1, 2, 3]:
        #rate of turn
        if len(binary_str) < 50:
            rot = 'No turn info available'
        else:
            rot= int(binary_str[42:50], 2)
            if rot == 0:
                rot = 'Not turning'
            elif (1 <= rot <= 126) or (-126 <= rot <= -1):
                rot = (abs(rot) / 4.733) ** 2
            elif rot == -127 or rot== 127:
                rot = 'Turn more than 5deg/sec'


        #speed over ground
        if len(binary_str) < 60:
            sog = float('nan')
        else:
            sog = int(binary_str[50:60], 2)*0.1
            if sog == 102.3:
                sog = 'SOG not available'
            elif sog == 102.2:
                sog = '102.2 knots or higher'

```

```
        #position accuracy
        if len(binary_str) < 61:
            position_acc = float('nan')
        else:
            position_acc = int(binary_str[60:61], 2)
            if position_acc == 1:
                position_acc = '<10m'
            elif position_acc ==0:
                position_acc = '>10m'


        #longitude
        if len(binary_str) < 89:
            long = float('nan')
        else:
            long = minute_to_dms(int(binary_str[61:89], 2))

        #latitude
        if len(binary_str) < 116:
            lat = float('nan')
        else:
            lat = minute_to_dms(int(binary_str[89:116], 2))

        #Course over ground
        if len(binary_str) < 128:
            cog = float('nan')
        else:
            cog = int(binary_str[116:128], 2)
            if cog == 3600:
                cog = 'NaN'

        #True Heading
        if len(binary_str) < 137:
            heading = float('nan')
        else:
            heading = int(binary_str[128:137], 2)
            if heading == 511:
                heading = 'Not Available'

        #Radio status
        if len(binary_str) < 168:
            radio_status = float('nan')
        else:
            radio_status = int(binary_str[149:168], 2)

        #Navigation status
        if len(binary_str) < 42:
            nav_status = float('nan')
        else:
            nav_status = get_navigation_status(int(binary_str[38:42],
    2))

        #Maneuver
        if len(binary_str) < 42:
            maneuver = get_maneuver_ind(0)
        else:
            maneuver = get_maneuver_ind(int(binary\_str[38:42], 2))
```

```

    ###############type 4 ###############

    elif message_type == 4:

        #longitude
        if len(binary_str) < 107:
            long = float('nan')
        else:
            long = minute_to_dms(int(binary_str[79:107], 2))

        #latitude
        if len(binary_str) < 134:
            lat = float('nan')
        else:
            lat = minute_to_dms(int(binary_str[107:134], 2))

        #Type of EPFD
        if len(binary_str) < 138:
            pos_fix_epfd = get_position_fix_type(0)
        else:
            pos_fix_epfd = get_position_fix_type(int(binary_str[134:138], 2))

        #SOTDMA state (radio)
        if len(binary_str) < 168:
            radio_status = float('nan')
        else:
            radio_status = int(binary_str[149:168], 2)

        #ETA
        if len(binary_str) < 78:
            eta = float('nan')
        else:
            eta = combine_to_datetime(binary_str)


    ###### type 5#####
    elif message_type == 5:

        #AIS version
        if len(binary_str) < 40:
            ais_version = float('nan')
        else:
            ais_version= int(binary_str[38:40],2)

        #IMO number
        if len(binary_str) < 70:
            imo = float('nan')
        else:
            imo= int(binary_str[40:70],2)

        #Call Sign
        if len(binary_str) < 112:
            call_sign = float('nan')
        else:
            call_sign= int(binary_str[70:112],2)
```

```
        #Vessel name
        if len(binary_str) < 232:
            vessel_name = float('nan')
        else:
            vessel_name= int(binary_str[112:232],2)

        #Ship type
        if len(binary_str) < 240:
            ship_type = get_ship_type(100)
        else:
            ship_type= get_ship_type(int(binary_str[232:240],2))

        #Dimension to Bow
        if len(binary_str) < 249:
            a = float('nan')
        else:
            a = int(binary_str[240:249], 2)

        #Dimension to Stern
        if len(binary_str) < 258:
            b = float('nan')
        else:
            b = int(binary_str[249:258], 2)

        #Dimension to Port
        if len(binary_str) < 264:
            d = float('nan')
        else:
            d = int(binary_str[258:264], 2)

        #Dimension to Starboard
        if len(binary_str) < 270:
            c = float('nan')
        else:
            c = int(binary_str[264:270], 2)

        #Position fix type
        if len(binary_str) < 274:
            pos_fix_epfd = get_position_fix_type(0)
        else:
            pos_fix_epfd = get_position_fix_type(int(binary_str
    [270:274], 2))

        #Draught
        if len(binary_str) < 302:
            draught = float('nan')
        else:
            draught = int(binary_str[294:302], 2)/10

        #Destination
        if len(binary_str) < 422:
            destination = float('nan')
        else:
            destination = int(binary_str[302:422], 2)

    return {
```

```
            'Timestamp': time,
            'Packet Type': packet_type.lstrip('!'),
            'Channel': channel,
            'Message Type': message_type,
            'MMSI': mmsi,
            'Navigation Status': nav_status,
            'Repeat Indicator':repeat_ind,
            'IMO': imo,
            'ROT': rot,
            'SOG': sog,
            'COG': cog,
            'Position Accuracy': position_acc,
            'Longitude':long,
            'Latitude':lat,
            'Vessel name': vessel_name,
            'Ship type': ship_type,
            'True Heading': heading,
            'Radio status': radio_status,
            'Destination': destination,
            'Maneuver Indicator': maneuver,
            'Draught': draught,
            'Position fix type': pos_fix_epfd,
            'Call sign': call_sign,
            'ETA': eta,
            'A':a,
            'B': b,
            'C': c,
            'D': d
        }
```

Listing 5: Python code for `decode_ais_message` function

4.6 `combine_to_datetime` Function

The `combine_to_datetime` function converts a segment of a binary string into a datetime object. It extracts specific date and time components (year, month, day, hour, minute, and second) from the binary string by decoding predefined bit positions into integers. These extracted components are then combined using the datetime constructor from the datetime library.

```
from datetime import datetime
def combine_to_datetime(binary_str):
    return datetime(year=int(binary_str[38:52], 2),
                    month=int(binary_str[52:56], 2),
                    day=int(binary_str[56:61], 2),
                    hour=int(binary_str[61:66], 2),
                    minute=int(binary_str[66:72], 2),
                    second=int(binary_str[72:78], 2))
```

Listing 6: Python code for `combine_to_datetime` function

4.7 `get_maneuver_ind` Function

This function translates a maneuver indicator from AIS data into a human-readable format. It uses a dictionary, maneuver_decode, to map integer values to descriptive strings based on the AIS standard. The function takes an integer maneuver as input and returns the corresponding description, such as "Not available (default)" or "No special maneuver". If the input value is not in the dictionary, it returns "Unknown status".

```python
def get_maneuver_ind(maneuver):
    # Define the navigation statuses based on AIS standard
    maneuver_decode = {
        0: "Not available (default)",
        1: "No special maneuver",
        2: "Special maneuver(such as regional passing arrangement)"
    }

    # Return the corresponding navigation status
    return maneuver_decode.get(maneuver, "Unknown status")
```

Listing 7: Python code for `get_maneuver_ind` function

4.8 `get_position_fix_type` Function

This function translates a position fix type from AIS data into a readable format. It uses again a dictionary, position_fix_decode, to map integer values to descriptive strings based on the AIS standard. The function takes an integer `pos_fix_epfd` as input and returns the corresponding description, such as "Undefined (default)", "Chayka" or "GPS". If the input value is not in the dictionary, it returns "Unknown status".

```python
def get_position_fix_type(pos_fix_epfd):
    # Define the navigation statuses based on AIS standard
    position_fix_decode = {
        0: "Undefined (default)",
        1: "GPS",
        2: "GLONASS",
        3: "Combined GPS/ GLONASS",
        4: "Loran-C",
        5: "Chayka",
        6: "Integrated navigation system",
        7: "Surveyed",
        8: "Galileo",
        9: "Reserved",
        10: "Reserved",
        11: "Reserved",
        12: "Reserved",
        13: "Reserved",
        14: "Reserved",
        15: "Internal GNSS"
    }
    # Return the corresponding navigation status
    return position_fix_decode.get(pos\_fix_epfd, "Unknown status")
```

Listing 8: Python code for `get_position_fix_type` function

4.9 minute_to_dms Function

This function decodes the binary string segment into DMS (degrees, minutes, seconds). It first divides the input value by 600000.0 to obtain the total minutes, as specified by the AIS standard. It then extracts the degrees as the integer part of the total minutes. The remaining fractional minutes are converted to minutes and seconds. Finally, the function returns a formatted string representing the coordinates in DMS format, such as "degrees minutes' seconds"". This function is useful for converting AIS positional data into a more readable geographic coordinate format.

```python
def minute_to_dms(value):
    # Convert from thousandths of a minute to minutes
    total_minutes = value / 600000.0

    # Extract the degrees, minutes, and seconds
    degrees = int(total\_minutes)
    minutes_decimal = (abs(total_minutes - degrees)) * 60
    minutes = int(minutes_decimal)
    seconds = (minutes_decimal - minutes) * 60

    return f"{degrees} {minutes}' {seconds:.1f}\""
```

Listing 9: Python code for get_position_fix_type function

4.10 get_ship_type Function

This function operates on the same principle as previously described. For a detailed explanation of its logic, please refer to the earlier section.

```python
def get_ship_type(ship_type):
    ship_type_decode = {
        0: "Not available (default)",
        1: "Reserved for future use",
        2: "Reserved for future use",
        3: "Reserved for future use",
        ...
        98: "Other Type, Reserved for future use",
        99: "Other Type, no additional information"
    }

    # Return the corresponding ship type
    return ship_type_decode.get(ship_type, "Unknown ship type")
```

Listing 10: Python code for get_ship_type function

4.11 get_navigation_status Function

This function operates on the same principle as previously described. For a detailed explanation of its logic, please refer to the earlier section.

```python
def get_navigation_status(nav_status):
    # Define the navigation statuses based on AIS standard
    nav_status_decode = {
```

```python
        0: "Underway using engine",
        1: "At anchor",
        2: "Not under command",
        3: "Restricted manoeuverability",
        4: "Constrained by her draught",
        5: "Moored",
        6: "Aground",
        7: "Engaged in fishing",
        8: "Underway sailing",
        9: "Reserved for future amendment of Navigational Status for HSC",
        10: "Reserved for future amendment of Navigational Status for WIG",
        11: "Power-driven vessel towing astern (regional use)",
        12: "Power-driven vessel pushing ahead or towing alongside (regional use)",
        13: "Reserved for future use",
        14: "AIS-SART is active",
        15: "Undefined (default)"
    }

    # Return the corresponding navigation status
    return nav_status_decode.get(nav_status, "Unknown status")
```

Listing 11: Python code for `get_navigation_status` function

5 Results

Add final version when finished

Timestamp	Packet Typ	Channel	Message T	MMSI	Navigation	Repeat Ind	IMO	ROT	SOG	COG	Position A	Longitude	Latitude	Vessel nam	Ship type	True Head	Radio statu	Destinatio	Maneuver	Draught	Position fi	Call sign	ETA	A	B	C	D	Error
17:51.0	AIVDM	A	5	6.04E+08	NaN	0	4.39E+08	No turn inf	0	Nan	Nan	0	0	6.75E+35	Passenger,	0	0		Not available (default		Undefined	2.23E+12		113	31	11	17	
17:51.0	BSVDM	A	1	3.53E+08	Underway	0	0	Not turnin	12.5	1728	>10m	152Â° 7' 1.	212Â° 0' 4	NaN	Not availal	170	81925	NaN	Not availal	0	Undefined	0		0	0	0	0	
17:51.3	BSVDO	A	4	2579991	NaN	0	0	No turn inf	0	Nan	Nan	152Â° 8' 7.	212Â° 26'	NaN	Not availal	0	114692	NaN	Not availal	0	Internal GI	0	###########	0	0	0	0	
																												Checksum mismatch
17:52.7	BSVDM	A	1	4.32E+08	Underway	0	0	Not turnin	11.3	3501	>10m	152Â° 6' 5	212Â° 21'	NaN	Not availal	350	2200	NaN	Not availal	0	Undefined	0		0	0	0	0	
																												Checksum mismatch
17:55.0	BSVDM	B	1	5.38E+08	Underway	0	0	2.187374	16.4	1400	>10m	151Â° 59'	213Â° 3' 5	NaN	Not availal	139	81928	NaN	Not availal	0	Undefined	0		0	0	0	0	
17:55.9	BSVDM	B	1	4.77E+08	Underway	0	0	0.401763	14.5	1320	>10m	151Â° 50'	213Â° 15'	NaN	Not availal	129	2257	NaN	Not availal	0	Undefined	0		0	0	0	0	
18:00.5	BSVDM	B	1	3.53E+08	Underway	0	0	Not turnin	12.5	1726	>10m	152Â° 7' 1	212Â° 0' 4	NaN	Not availal	170	2289	NaN	Not availal	0	Undefined	0		0	0	0	0	
18:01.2	BSVDO	B	4	2579991	NaN	0	0	No turn inf	0	Nan	Nan	152Â° 8' 7.	212Â° 26'	NaN	Not availal	0	114692	NaN	Not availal	0	Internal GI	0	###########	0	0	0	0	
18:01.8	BSVDM	A	3	5.38E+08	Underway	0	0	1.116007	16.4	1401	>10m	151Â° 59'	213Â° 3' 5	NaN	Not availal	139	24355	NaN	Not availal	0	Undefined	0		0	0	0	0	
18:02.5	BSVDM	B	1	4.32E+08	Underway	0	0	Not turnin	11.3	3504	>10m	152Â° 6' 5	212Â° 21'	NaN	Not availal	350	20016	NaN	Not availal	0	Undefined	0		0	0	0	0	
																												Checksum mismatch
18:08.2	BSVDM	B	1	4.77E+08	Underway	0	0	Not turnin	14.5	1322	>10m	151Â° 50'	213Â° 15'	NaN	Not availal	129	49158	NaN	Not availal	0	Undefined	0		0	0	0	0	
18:11.2	BSVDO	A	4	2579991	NaN	0	0	No turn inf	0	Nan	Nan	152Â° 8' 7.	212Â° 26'	NaN	Not availal	0	114692	NaN	Not availal	0	Internal GI	0	###########	0	0	0	0	
18:11.3	BSVDM	A	1	3.53E+08	Underway	0	0	Not turnin	12.5	1725	>10m	152Â° 7' 1	212Â° 0' 4	NaN	Not availal	170	114693	NaN	Not availal	0	Undefined	0		0	0	0	0	
18:12.8	BSVDM	A	1	4.32E+08	Underway	0	0	Not turnin	11.3	3502	>10m	152Â° 6' 5	212Â° 21'	NaN	Not availal	350	67101	NaN	Not availal	0	Undefined	0		0	0	0	0	
																												Checksum mismatch
18:13.8	BSVDM	A	1	5.38E+08	Underway	0	0	Not turnin	16.4	1400	>10m	151Â° 59'	213Â° 3' 5	NaN	Not availal	139	20016	NaN	Not availal	0	Undefined	0		0	0	0	0	
18:14.0	BSVDM	A	1	4.77E+08	Underway	0	0	Not turnin	14.5	1323	>10m	151Â° 50'	213Â° 15'	NaN	Not availal	129	34380	NaN	Not availal	0	Undefined	0		0	0	0	0	
18:21.2	BSVDO	B	4	2579991	NaN	0	0	No turn inf	0	Nan	Nan	152Â° 8' 7.	212Â° 26'	NaN	Not availal	0	114692	NaN	Not availal	0	Internal GI	0	###########	0	0	0	0	
18:21.3	BSVDM	B	1	4.32E+08	Underway	0	0	Not turnin	11.3	3497	>10m	152Â° 6' 5	212Â° 21'	NaN	Not availal	351	20016	NaN	Not availal	0	Undefined	0		0	0	0	0	
18:22.0	BSVDM	B	1	3.53E+08	Underway	0	0	Not turnin	12.4	1721	>10m	152Â° 7' 1	212Â° 0' 4	NaN	Not availal	170	81925	NaN	Not availal	0	Undefined	0		0	0	0	0	
																												Checksum mismatch

Figure 2: Resulting AIS message Dataset

6 Limitations

Add when finished

7 Conclusion

Add when finished

8 References

Danish Maritime Authority - AIS. Retrieved from:
`https://github.com/dma-ais`
Automatic identification system. Retrieved from:
`https://en.wikipedia.org/wiki/Automatic_identification_system`
AIS data Danish Maritime Authority. Retrieved from:
`https://www.dma.dk/safety-at-sea/navigational-information/ais-data`
GH AIS Message Format. Retrieved from:
`https://www.iala-aism.org/wiki/iwrap/index.php/GH_AIS_Message_Format`
AIVDM/AIVDO protocol decoding. Retrieved from:
`https://gpsd.gitlab.io/gpsd/AIVDM.html`
pyais. Retrieved from:
`https://github.com/M0r13n/pyais/tree/master?tab=readme-ov-file`
ais-protocol-decoding. Retrieved from:
`https://github.com/doron2402/ais-protocol-decoding`